Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matevž Žugelj

Grafični prikazovalnik podatkov v realnem času

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Del kontrolnega sistema za pospeševalnik delcev je vizualizacija procesov na uporabniškem vmesniku za končne uporabnike. Uporabniški vmesnik izdelamo z uporabo sistema SCADA, na njem pa prikažemo stanja procesov in naprav pomembnih za pospeševalnik. Ko pri izmenjavi podatkov z napravami, ki jih uporabimo za vizualizacijo stanj procesov, naletimo na komunikacijski protokol, ki ga sistem SCADA ne pozna, je naša naloga, da razvijemo rešitev, ki bo protokol razumela in vizualizirala prejete podatke. V diplomski nalogi izdelajte grafični prikazovalnik podatkov v realnem času kot rešitev specifičnega problema.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
SCADA	Supervisory Control And Data Acquisition	nadzor in krmiljenje tehnoloških procesov
DDS	Data Distribution Service	storitev za prenos podatkov
GUI	Graphical User Interface	grafični uporabniški vmesnik
HMI	Human-Machine Interface	strojno-uporabniški vmesnik

Povzetek

Del kontrolnega sistema za pospeševalnik delcev je vizualizacija procesov na uporabniškem vmesniku za končne uporabnike. Uporabniški vmesnik izdelamo z uporabo sistema SCADA, na njem pa prikažemo stanja procesov in naprav pomembnih za pospeševalnik. Ko pri izmenjavi podatkov z napravami, ki jih uporabimo za vizualizacijo stanj procesov, naletimo na komunikacijski protokol, ki ga sistem SCADA ne pozna, je naša naloga, da razvijemo rešitev, ki bo protokol razumela in vizualizirala prejete podatke. Za specifičen problem je rešitev grafični prikazovalnik podatkov v realnem času. Ob izdelavi aplikacije opišemo celoten razvojni proces od predstavitve širše slike pospeševalnikov delcev do problema, zahtev, načrtovanja, implementacije in testiranja.

Ključne besede: Graf, Qt, Qwt, WinCC OA.

Abstract

Part of the particle accelerator control system is visualization of processes on a graphical user interface (abbr. *GUI*) for end users. GUI is made by using a SCADA system and displays states of processes and devices important to the particle accelerator. When encountered with a non-supported communication protocol and which content needs to be visualized, it is our task to develop a solution which understands the protocol and visualizes the content. For a specific problem a solution for it is a realtime graphical data viewer. In addition to making the application, a detailed description of the whole development process is given, from the big picture of particle accelerators to the specifications and requirements, architecture design, implementation and testing.

Keywords: Graph, Qt, Qwt, WinCC OA.

Poglavje 1

Uvod

Živimo v svetu polnem informacij, ki jih dobimo s filtriranjem, oblikovanjem in agregiranjem podatkov. Informacije si interpretiramo, se na podlagi njih odločamo in ukrepamo, ob tem pa se učimo in si pridobivamo znanje. V računalništvu je veliko vizualnih informacij. Že takoj ob zagonu osebnega računalnika nas razveseli novica, da so za naš operacijski sistem in ostalo programsko opremo na voljo posodobitve. Kar se zgodi pred tem je to, da računalnik primerja različice nameščenih programov z najnovejšimi in ugotovi, da obstajajo novejše različice. Računalnik torej zbira, filtrira in agregira podatke in iz njih izlušči informacijo o tem, da so na voljo posodobitve. Podobno je pri kontrolnemu sistemu za pospeševalnik delcev, kjer je nešteto kosov senzorike, ki izmerjene vrednosti različnih fizikalnih količin posredujejo raznim odjemalcem kot so različne naprave ali druge krmilne in kontrolne komponente. Odjemalci so tudi aplikacijska ogrodja, ki razumejo standardizirane in nestandardizirane industrijske komunikacijske protokole, nam ponujajo enoten vmesnik med njimi in nam omogočajo, da vse te podatke združimo in predstavimo uporabniku na grafičnem vmesniku v obliki informacij. Te predstavimo uporabniku na način, ki mu najbolj odgovarja in ga najlažje razume. Grafični vmesnik je lahko sestavljen iz enostavnih komponent kot so slike, vnosna in prkazna polja, seznami in tabele ali pa iz kompleksnejših komponent kot so urejevalniki slik, besedil in prikazovalniki

multimedijskih vsebin. Nekatera ogrodja omogočajo razvoj svojih lastnih komponent, v primeru da želimo podpreti pridobivanje podatkov iz storitev, ki uporabljajo nestandardne komunikacijske protokole ali pa želimo prikazati podatke na druge načine kot nam jih ponujajo ogrodja. Grafični prikazovalnikom podatkov v realnem času je razvit kot razširitev grafičenga vmesnika. Ta podatke pridobiva iz zunanje storitve, ki uporablja nestandarden komunikacijski protokol, in jih prikazuje v obliki krivulj v dvodimenzionalnem prostoru.

Poglavje 2

Pospeševalnik delcev

Pospeševalnik delcev je naprava, ki uporablja električno polje za pospeševanje električno nabitih delcev na hitrost blizu svetlobne hitrosti [18]. Pospešeni delci so združeni v žarke (angl. *beams*), ki se jih uporablja za različne namene. Obstajata dve vrsti pospeševalnikov, glede na način pospeševanja. Elektrostatični pospeševalniki delcev uporabljajo električna polja za pospeševanje. Primer teh pospeševalnikov delcev so starejši modeli televizorjev, ki za pospeševanje eletkronov oziroma izris slike uporabljajo katodno cev. Končna hitrost delcev pospeševanih s to metodo je omejena s pojavom električnega obloka (angl. *electrical breakdown*), z razliko od pospeševalnikov, ki za pospeševanje uporabljajo radiofrekvenčna elektromagnetna (oscilirajoča) polja. To metodo pospeševanja uporablja večina današnjih večjih pospeševalnikov.

2.1 Namen in področja uporabe

Pospeševalniki delcev se uporabljajo za raziskave v znanosti, številnih tehničnih in industrijskih področjih. Nekateri se uporabljajo za izvajanje eksperimentov teorije delcev, ki razširjajo obzorja o dinamiki in strukturi materije, prostora in časa. Drugi izkoroščajo sevanje, ki ga oddajajo pospeševani delci za raziskovanje strukture atomov, kemije, fizike, biologije in tehnologije. Obstajajo tudi medicinski pospeševalniki delcev, s katerimi obsevajo rakava

področja.

2.2 Pospeševanje z oscilirajočimi polji

Z uporabo oscilirajočih polj se torej izognemo elektičnemo obloku, kar omogoča pospeševanje delcev do večjih energij. Pri tej metodi ponavadi uporabljamo več pospeševalnih struktur v pospeševalniku. Te so lahko razporejene v ravni liniji, kjer govorimo o linearnem pospeševalniku [14] (angl. *linear accelerator* okr. *LINAC*), in v krogu, kjer govorimo o cikličnem pospeševalniku [6] (angl. *cyclic accelerator* okr. *cyclotron*) oziroma ciklotronu. Izbira tipa pospeševalnika je odvisna od namena uporabe.
Moderni pospeševalniki za pospeševanje uporabljajo radiofrekvenčne votline [26] (angl. *radio frequency cavities*, okr. *RF cavities*). Te pospešjejo skupine delcev (angl. *particle bunches*). Pospešeni delci so na koncu pospeševalnika v obliki skoncentriranega žarka (angl. *beam*), kjer so delci čim bolj tesno en ob drugem in potujejo v čim bolj v ravni liniji. Ker je po naravi nemogoče usmeriti dva delcev v enako smer, se delci pri potovanju skozi prostor sčasoma razpršijo. Pospeševalnik zato med pospeševanjem delce ponovno fokusira v skoncentriran žarek. Naprave, ki za to skrbijo se imenujejo magnetne leče, ki so sestavljene iz kvadrupolnih [25] in sextupolnih [29] magnetov.

2.3 Kroženje delcev

Ciklični pospeševalniki poleg pospeševanja in fokusiranja delcev skrbijo še za kroženje delcev, saj se ti gibljejo v krožnih, elipsastih ali spiralnih poteh. Kroženje delcev dosežejo z uporabo usmerjevalnih oziroma dipolnih magnetov, ki spreminjajo oziroma ukrivljajo pot delca in ne njegove hitrosti, saj so magnetne silnice zmeraj pravokotne na vektor hitrosti. V ciklotronu je magnetno polje konstantno in ni časovno odvisno, zato se delci med pospeševanjem gibljejo v obliki spirale. Naslednik ciklotrona, sinhrotron [31] (angl. *synchrotron)*, ima krmilno magnetno polje, ki je časovno sorazmerno

odvisno od naraščajoče kinetične energije delcev, zato delci potujejo v obliki elipse.

2.4 LINAC ali ciklotron?

Prednosti linearnega pospeševalnika so v njegovi sekvenčni modularni strukturi, prednosti ciklotrona pa v njegovi kompaktnosti [13]. Vsak ima svoje prednosti in slabosti, izbira pa je odvisna od namena uporabe.
Ciklotron je s svojo ciklično strukturo kompakten in stroškovno učinkovit. Manjka mu redundanca, ki je ključna za toleriranje in odpravlanje napak, zato ciklotron ne dosega višjih odstotkov dosegljivosti (angl. *availability*). Prav tako je nadgradnja moči zanj zelo težko izvedljiva.
Linearni pospeševalnik je lahko sestavljen iz večih neodvisnih pospeševalnih struktur. S tem je stopnja modularnosti visoka, kar nam zagotovi redundanco in posledično visoke odstotke dosegljivosti. Nadgradnja energije končnega žarka je mogoča s preprostim dodajanjem pospeševalnih struktur.

Ključne točke splošne primerjave obeh tipov pospeševalnikov so naštete v Tabeli 2.1.

LINAC	**CIKLOTRON**
Zahteva veliko prostora in je lahek	Je kompakten in težek
Za izgradnjo potrebnih veliko sredstev	Cenejši za izgradnjo
Energijsko manj učinkovit	Energijsko bolj učinkovit
Modularnost poskrbi za redundanco	Ni redundance
Nadgradnja energije preprosta	Nadgradnja energije zahtevna
Ekstrakcija žarka enostavna	Ekstrakcija žarka zahtevna in vključuje izgube
Visok tok žarka (100 mA)	Nizek tok žarka (5 mA)

Tabela 2.1: Splošna primerjava linearnih in cikličnih pospeševalnikov.

2.5 Splošna zgradba sinhrotrona

Sinhrontron je sestavljen iz različnih naprav, ki jih lahko razdelimo v nekaj večjih komponent [36]. Če opisujemo sinhrotron od izvora do ponora začnemo z linearnim pospeševalnikom. Naloga tega je da delce iz izvora pospeši na začetno hitrost. Delce nato preko prenosne linije prenesemo v sinhrotron. Prenosne linije so sestavljene iz magnetnih leč (glej 2.2) in usmerjevalnih magnetov (glej 2.3), da delci prispejo iz ene točke v drugo brez razprševanja. Ko delci prispejo v sinhrotron te dodatno pospešimo na končno kinetično energijo. Pospešene delce nato preko druge prenosne linije prenesemo v shranjevalni obroč (angl. *storage ring*). Tam delci krožijo dokler jih ne uporabijo, med tem pa počasi izgubljajo energijo. Njihova uporaba je še zadnji korak, ko jih iz shranjevalnega obroča izločimo v žarčne linije (angl. *beam line*), od tod naprej pa jih uporabljamo za različne namene.

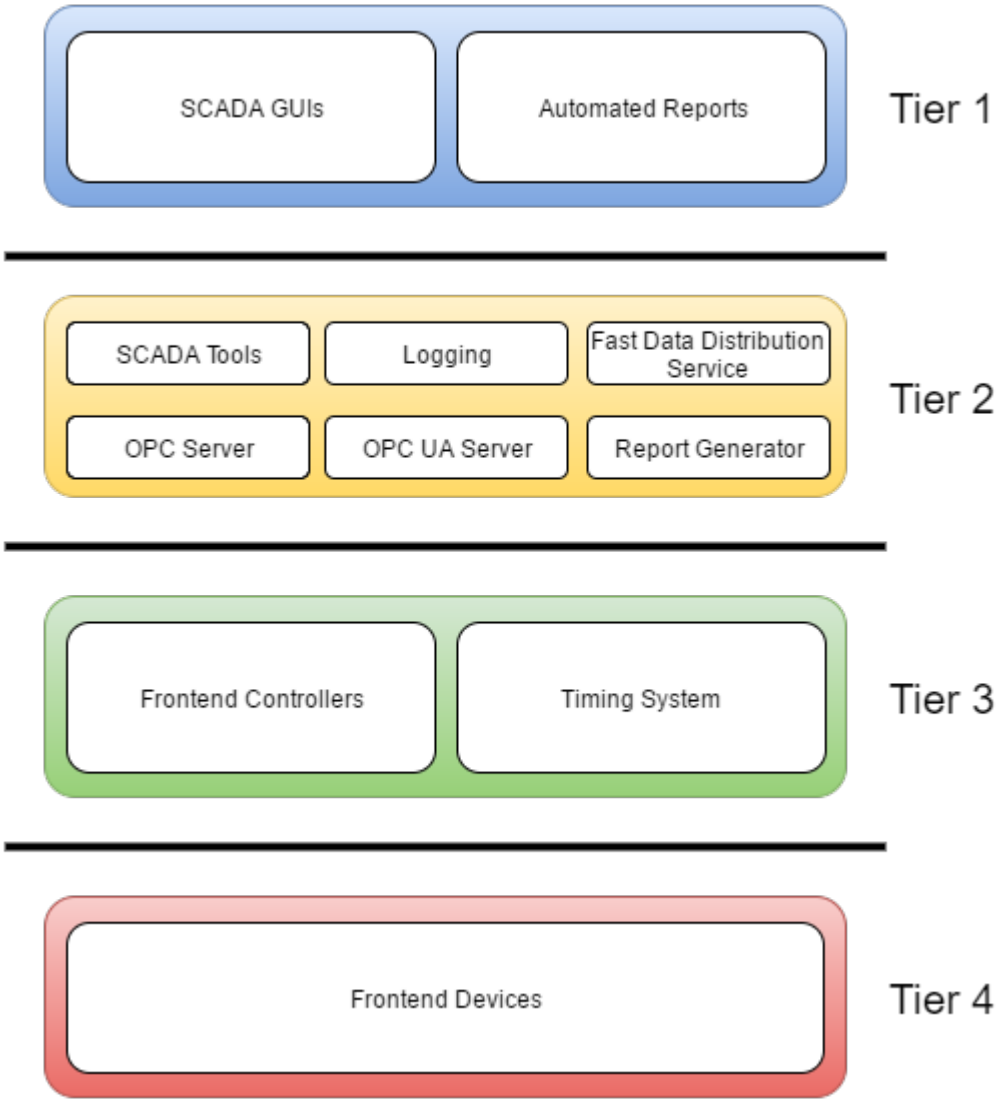
Poglavje 3

Kontrolni sistem

Arhitekturo kontrolnega sistema pospeševalnika delcev najlažje opišemo skozi industrijski model, sestavljen iz štirih nivojev [11]:

- Predstavitvena plast (angl. *presentation tier*)
- Procesna plast (angl. *processing tier*)
- Plast opreme (angl. *equipment tier*)
- Plast končnih naprav (angl. *frontend tier*)

Komponente na različnih plasteh (glej sliko 3.1) za medsebojno komunikacijo uporabljajo ethernet povezavo. Poleg ethernet povezave lahko komunikacija med 3. in 4. plastjo poteka tudi preko vmesnikov fieldbus in vmesnikov drugih vrst.



Slika 3.1: Razporeditev komponent kontrolnega sistema po 4 plastnem industrijskem modelu (poenostavitev slike iz [11]).

3.1 Plast končnih naprav

Plast končnih naprav zajema naprave, ki krmilijo strojno opremo pospeševalnika. Sem spadajo naprave, ki ponujajo nizkonivojske knjižnice API (angl. *application programming interface*) za operacijske sisteme, ki operacije izvajajo v realnem času. Nizkonivojske knjižnice API se na plasti opreme uporabljajo za implementacijo krmilnikov (angl. *front end controller*, okr. *FEC*). Na plast končnih naprav spada tudi programabilni logični kontroler (angl. *programmable logic controller*, okr. *PLC*), ki lahko krmili več naprav (magneti, pretvorniki napetosti, ...) hkrati.

3.2 Plast opreme

Plast opreme služi dvema namenoma: Prvi je poenotenje različnih tipov naprav z implementacijo krmilnikov, ki uporabljajo enak načrtovalski vzorec. Drugi je lokalno shranjevanje parametrov za časovno konfiguracijo naprav.

3.2.1 Časovni sistem (angl. *timing system*)

Naprave v sinrotronu, ki vplivajo na žarek, krmili časovni sistem, ki je del 3. plasti. Ker se delci v pospeševalniku gibljejo blizu svetlobne hitrosti, adaptivno krmiljenje teh naprav ni mogoče, saj bi se odzivni časi gibali v rangu nano in pikosekund, kar pa je daleč izven dosega odzivnih časov elektronike. Zato se naprave krmili z naprej določenimi sekvencami ukazov preko časovnega sistema [28] (angl. *timing system*), kjer sekvence navodil oziroma dogodke (angl. *events*) napravam oziroma prejemnikom dogodkov (angl. *event receivers*) oddaja generator dogodkov (angl. *event generator*). Tako naprave vedo kakšna bo njihova nasavitev v vsaki časovni točki pospeševalnega cikla. Pri tem je pomembno, da so naprave med seboj popolnoma sinhronizirane in da ne pride do odstopanj.

3.3 Procesna plast

Procesni plast konfigurira in nadzira 3. in 4. plast. Komponente na tej plasti skrbijo za:

- virtualizacijo vseh fizičnih naprav v strukture podatkovnih točk (angl. *data points*) in uporaba enotnega avtomata (angl. *state machine*) za naprave, ki delujejo kot avtomati,
- izdelavo enotne slike sistema s predstavitvijo vseh naprav, strojev in programske opreme v podatkovnih točkah,
- uveljavitev nadzora dostopa za različne vloge,
- nastavitve krmilnikov FEC preko protokolov OPC, OPC UA, HTTP in FTP,
- shranjevanje sistemske konfiguracije v podatkovnih zbirkah,
- sinhronizacijo in regulacijo dostopa do skupnih virov,
- rokovanje z alarmi in napakami, zbiranje zabeleženih sporočil,
- distribucija pridobljenih procesnih podatkov,
- generiranje poročil,
- avtomatsko izvajanje procedur za nadzor in krmiljenje naprav.

3.3.1 OPC in OPC UA

Open Platform Communications [16] (okr. *OPC*) in njegov naslednjik Open Platform Communications Unified Architecture [17] (okr. *OPC UA*) sta zbirki standardov, ki definirata način industrijske komunikacije za nadzor procesov. OPC in OPC UA omogočata branje in pisanje podatkov v realnem času, dostop do zgodovinskih vrednosti podatkov in dostop do informacij o alarmih in dogodkih. OPC UA je naslednjih OPC. Ima več funkcionalnosti, je bolj generičen, skalabilen in dostopen na večih platformah.

3.3.2 Sistemi SCADA

Glavna komponenta na tej plasti je sistem SCADA, ki skrbi za nadzor in konfiguracijo naprav, zbiranje in procesiranje podatkov. Sistem SCADA omogoča interakcijo med fizičnimi napravami in končnimi uporabniki preko uporabniško-strojnih vmesnikov [8] (angl. *human-machine interface* okr. *HMI*) oziroma grafičnih uporabniških vmesnikov (angl. *graphical user interface*, okr. *GUI*) na 1. plasti.

3.4 Predstavitvena plast

Na predstavitveni plasti predstavimo podatke končnemu uporabniku na grafičnem uporabniškem vmesniku, ki ga izdelamo z uporabo orodij za izdelavo HMI, ki so del sistema SCADA.

Poglavje 4

Specifikacija in analiza zahtev

4.1 Problem in specifikacija zahtev

Na začetku vsake rešitve je problem, ki stoji med začetno točko (kjer stojimo) in končno točko (kamor želimo priti). Iz začetne v končno točko pridemo na način, ki ga določa rešitev, s katerim obidemo ali odpravimo problem. Za izdelavo rešitve moramo torej dobro poznati problem, da vemo kaj rešitev sploh bo, zato iz opisa problema specificiramo zahteve.

Realni problem vsebuje pospeševalnik delcev, kjer za prikaz podatkov iz naprav pospeševalnika na grafičnem vmesniku uporabljamo sistem SCADA z imenom WinCC Open Architecture [9] (okr. *WinCC OA*) različice 3.12. Del podatkov, ki jih želimo prikazati, so meritve fizikalnih količin, ki se spreminjajo skozi čas. Te prihajajo preko storitve za prenos podatkov (angl. *Data Distribution Service*, okr. *DDS*) iz procesne plasti. Storitev DDS za komunikacijo uporablja nestandardni protokol TCP/IP, ki ga WinCC OA ne razume. Naša naloga je prikazati meritve prejete od storitve DDS, nestandardni protokol TCP/IP pa predstavlja problem.

Rešitev naj bo aplikacija, ki prikazuje podatke (meritve) prejete od storitve DDS v obliki krivulj sestavljenih iz točk (ena točka ima koordinati x in y) v dvodimenzionalnem prostoru. Podatki naj se prikazujejo v realnem času s hitrostjo vsaj 100 000 točk na sekundo. Rešitev naj omogoča prejemanje

samo tistih podatkov od storitve DDS, ki jih potrebujemo (na nove podatke se naročamo ali se odjavljamo od obstoječih naročnin). Rešitev naj omogoča prikazovanje podatkov prejetih s strani WinCC OA, kot omogočajo obstoječi grafični elementi vključeni v WinCC OA. Podatke želimo na uporabnikovo zahtevo v realnem času tudi shranjevati v datoteko. Rešitev naj vsebuje oznake in imena osi, naslov in legendo, s katero prikazujemo ali skrivamo obstoječe krivulje. Uporabnik se lahko premika po kooridnatnih oseh v vse smeri in povečuje ali zmanjšuje ločljivost. Podatki naj se prikazujejo na dva načina. Pri prvem se obstoječi podatki (prejeti pod določeno naročnino) zamenjajo z novimi, pri drugem pa naj se pripnejo obstoječim. Za način pripernjanja naj obstaja omejitev v številu sekund, ki jo poda uporabnik. Število sekund predstavlja časovno omejitev, podatki starejši od te omejitve pa se zbrišejo.

Vse naštete zahteve so strnjene in z oznakami zbrane v Tabeli 4.1.

Zahteva	**Opis**
REQ-1	Aplikacija prikazuje podatke prejete od storitve DDS.
REQ-2	Aplikacija naj se uporablja kot del grafičnega vmesnika v sistema SCADA imenovanega WinCC OA verzije 3.12.
REQ-3	Aplikacija prikazuje podatke kot krivulje, sestavljene iz točk. Krivulje so prikazane v dveh dimenzijah na grafu.
REQ-4	Aplikacija prikazuje podatke prejete od WinCC OA.
REQ-5	Aplikacija omogoča upravljanje naročnin, pod katerimi prihajajo podatki.
REQ-6	Aplikacija prikazuje vsaj 100 000 točk na sekundo v realnem času.
REQ-7	Aplikacija omogoča shranjevanje prikazanih podatkov v datoteko na trdi disk v obliki formata csv [5].
REQ-8	V datoteko na trdi disk shranjujemo točke v realnem času.
REQ-9	Aplikacija prikazuje imena osi in merske enote.
REQ-10	Aplikacija prikazuje naslov grafa.
REQ-11	Aplikacija vsebuje legendo z imeni krivulj. Z legendo lahko poljubne krivulje skrijemo ali prikažemo.
REQ-12	Aplikacija omogoča pripenjanje novih (prejetih) podatkov k obstoječim ali zamenjavo obstoječih podatkov z novimi (privzeto).
REQ-13	Uporabnik se lahko premika po koordinatnih oseh v vse smeri in povečuje ali zmanjšuje ločljivost.

Tabela 4.1: Seznam zahtev z opisi.

4.2 Analiza zahtev in načrtovanje

Zahteve znane, zato začnemo razmišljati kaj bo rešitev, kako jo bomo naredili in če je to sploh možno. Izbrati je potrebno ustrezne tehnologije, ki jih bomo uprabili pri implementaciji rešitve. Skozi postopek analize in načrtovanja nas vodi specifikacija zahtev v tabeli 4.1, na katero se sklicujemo pri izboru tehnologij.

4.2.1 Storitev DDS

Zahteva REQ-1 pravi, da rešitev prikazuje podatke prejete od storitve DDS, ki nam zagotavlja visoke hitrosti prenosa podatkov v nasprotju z industrijskima standardoma OPC in OPC UA. To je glavni razlog za uporabo storitve DDS namesto industrijskih standardov OPC ali OPC UA.
Storitev DDS za prenos podatkov uporablja svoj protokol TCP/IP. Njegova arhitektura temelji na načrtovalskem vzorcu (angl. *design pattern*) opazovalca [10] (angl. *observer*) oziroma vsebinsko orientiranega založnika-naročnika [19] (angl. *content based publish-subscribe*).

Načrtovalski vzorci

Načrtovalski vzorci za razvoj programske opreme [7] (angl. *software design patterns*) ponujajo splošne rešitve za pogoste probleme pri razvoju programske opreme. Načrtovalski vzorci so formalizirani zapisi najboljših praks rešitev problemov. Opisujejo problem in način reševanja. Ne vsebujejo programske kode, le znanje, ki ga lahko prenesemo v poljuben programski jezik.

Vsebinsko orientiran založnik-naročnik

Vzorec vsebinsko orientiranega založnika-naročnika govori o problemu iz sveta računalništva skozi analogijo iz realnega sveta. Analogija se začne z založniki, ki objavljajo oziroma izdajajo vsebino (knjige, časopise oziroma podatke) pod določenimi naročninami (lahko tudi temami ali oznakami). Na drugi strani so naročniki, ki sklepajo naročnine ali se od njih odjavljajo.

Naročniki dobivajo vsebino za vsako sklenjeno naročnino, od trenutka naprej, ko je naročnina sklenjena. Naročniki ne prejemajo izdanih vsebin, ki so bile izdane pred sklenitvijo naročnine.

Implementacija vzorca in delovanje

Implementacija načrtovalskega vzorca vsebuje za distribucijo podatkov med založniki in naročniki še vmesni (posredni) del, ki se izvaja ločeno na strežniku v obliki storitve. Tako založniki in naročniki (aplikacije) med seboj komunicirajo posredno. Naročniki se na strežniku naročijo na določeno temo in prejemajo vsebino poslano pod to temo. Vsebine pod različnimi temami na strežnik pošiljajo založniki.
Storitev DDS pošilja podatke v poljubnem formatu, ki je vnaprej definiran. Za prikazovanje podatov v aplikaciji je uporabljen format, ki med drugim vsebuje naslednjo vsebino:

- Naslov teme pod katero so podatki objavljeni (REQ-11);
- Poljubno število krivulj s točkami (REQ-3);
- Imeni in merski enoti abscisne in ordinatne osi (REQ-9).

Storitev DDS ponuja API, napisan v programskem jeziku C++, ki vsebuje naročnika in naročnika na storitev DDS. Naročnik, ki prejema podatke storitve DDS in razume uporabljen komunikacijski protokol, izpolnjuje zahtevo REQ-1, zato bo vključen v aplikacijo.

4.2.2 WinCC OA

WinCC OA je zaprtokodna impementacija sistema SCADA. Uporablja se za nadzor različnih avtomatiziranih industrijskih procesov, v tem primeru delovanja in uporabe pospeševalnika delcev. WinCC OA ponuja funkcionalnosti kot so zbiranje in vizualizacija podatkov, arhiviranje podatkov, rokovanje z alarmi, beleženje dogodkov, distribucija sistema in sistemska redundanca

večih načinov. WinCC OA je modularen sistem, zato je vsaka večja funkcionalnost svoj proces. WinCC OA procese, ki skrbijo za funkcionalnosti, imenuje managerji (manager za prikazovanje GUI-jev, arhiviranje, rokovanje z dogodki, ...). Managerji med seboj komunicirajo in si izmenjujejo podatke.

Zbiranje podatkov

Podatki v WinCC OA pritekajo iz različnih virov, ki uporabljajo različne komunikacijske protokole enkapsulirane v industrijske standarde, kot sta OPC in OPC UA. Podatki se iz različnih virov zbirajo v posebno podatkovno strukturo, ki predstavlja podatkovno bazo sistema WinCC OA. Pomembno je, da so podatki zbrani tako, da do njih dostopamo na enak način, ne glede na njihov izvor. Podatki so zbrani v drevesni strukturi iz podatkovnih točk (angl. *Data Points*), ki jo definira WinCC OA. Sestavljena iz treh delov:

1. Tip podatkovne točke (angl. *Data Point Type*, okr. *DPT*). DPT vsebuje seznam vseh lastnosti, ki si jih pomnimo;
2. Podatkovna točka (angl. *Data Point*, okr. *DP*). DP je instanca DPT in ima svojo kopijo lastnosti, ki jih določa DPT;
3. Element podatkovne točke (angl. *Data Point Element*, okr. *DPE*). DP ima en DPE za vsako lastnost, ki jo določa DPT.

Prikazovanje podatkov

V celotnem od krmiljenja pospeševalnika je prikaz podatkov končnemu uporabniku edina stopnja, ki jo neposredno zazna. Orodje, s katerim nam WinCC omogoča izgradnjo grafičnega vmesnika, je grafični urejevalnik (angl. *graphics editor*) imenovan gedi.
Gedi omogoča izdelavo grafičnega vmesnika, na katerem z osnovnimi grafičnimi elementi kot so kvadrati, krogi in vnosna polja ali s kompleksnimi grafičnimi elementi kot so seznami in tabele, prikazujemo podatke iz drevesne podatkovne strukture končnim uporabnikom na grafičnem uporabniškem vmesniku.

4.2.3 Qt

Grafični elementi uporabljeni v orodju gedi so razviti v aplikacijskem ogrodju Qt [4, 24] različice 4.8. Qt je aplikacijsko ogrodje, podprto na večih platformah kot so Windows, Linux, Android, OS X in iOS. Qt je napisan v programskem jeziku C++.
V orodju gedi lahko na grafični vmesnik dodajamo ali odstranjujemo grafične elemente, jim določamo izgled in obnašanje. Zato moramo znati z njimi komunicirati.

Sistem signalov in rež

Ogrodje Qt uporablja sistem signalov in rež [22] (angl. *signal-slot system*), ki omogoča komunikacijo med objekti (grafični elementi). Qt se od drugih tovrstnih ogrodij najbolj razlikuje ravno po načinu komunikacije med objekti. Nekatera druga ogrodja uporabljajo sistem povratnih klicev (angl. *callbacks*), ki ne zagotavlja tipne varnosti (angl. *type safety*) in proženja povratnega klica s specificiranimi argumenti. Qtjev sistem signalov in rež uporablja tehniko tipne introspekcije [12], ki ponuja informacije o tipu in lastnostih objekta med delovanjem programa. Ta način komunikacije zagotavlja tipno varnost in proženje povratnih klicev s specificiranimi argumenti. Komunikacija med objekti je s sistemom signalov in rež je glavna lastnost ogrodja Qt.
WinCC OA nam ponuja različne načine s katerimi lahko definiramo grafičnemu elementu izgled in obnašanje. Prvi način določa obnašanje objekta v času izvajanja. To določajo kontorlne skripte (angl. *CTRL scripts*) napisane v skriptnem jeziku WinCC OA, ki je podoben programskemu jeziku ANSI C. Kontrolne skripte pišemo v orodju gedi. Izvajajo se ob dogodkih kot so inicializacija in deinicializacija objekta, pritiski miških gumbov, pritiski gumbov na tipkovnici in drugi. Za izvajanje skript skrbi tolmač (angl. *interpreter*) sistema WinCC OA, ki v izvajalnem času komunicira s Qt objektom. Drug način določa objektu lastnosti kot so barve ozadja in objekta, velikost, pozicijo na zaslonu, velikost in tip pisave. Gre za sistem oziroma vmesnik

imenovan Q_PROPERTY [21] in sistem slogovnih razpredelnic [23] (angl. *stylesheets*), ki definira izgled objektov. Slogovne razpredelnice so izpeljane iz kaskadnih slogovnih razpredelnic [3] (angl. *cascading style sheets*, okr. *CSS*), prilagojenim svetu Qt. V sistemu WinCC OA slogovne razpredelnice podamo preko samostojne datoteke s končnico .css ali preko sistema Q_PROPERTY.

Uporaba Qt v aplikaciji

Vsi obstoječi grafični elementi v WinCC OA črpajo podatke iz enotne drevesne strukture, mi pa želimo prejemati podatke iz storitve DDS kot pravi REQ-1. Zato izdelamo svoj grafični element z uporabo Qt, ki podpira zgoraj naštete načine definicije stila in obnašanja in ga tako lahko vgradimo uporabimo v WinCC OA, saj bo ta z njim znal komunicirati. Z dodajanjem obstoječih grafičnih elementov in z manipulacijo aplikacije preko kontrolnih skript podpremo zahteve, ki specificirajo spreminjaje delovanja aplikacije v izvajalnem času (REQ-4, REQ-5 in REQ-7), s sistemom Q_PROPERTY podpremo zahteve, ki ne spreminjajo delovanja aplikacije v realnem času (REQ-10) in z uporabo slogovnih razpredelnic definiramo vizualni slog.

Izdelana bo Qt aplikacija, ki zna komunicirati s storitvijo DDS in WinCC OA in ga implementiramo z uporabo in dedovanjem razreda QWidget, ki je del knjižnice Qt. Razred QWidget omogoča definicijo dodatnih lastnosti objektov (sistem Q_PROPERTY). Za definiranje stila s slogovnimi razpredelnicami je potrebno spremeniti vsebino zapisa razpredelnic, saj za izrisovanje skrbi samo ogrodje Qt. Izvajanje skript ob poljubnih dogodkih je možno z uporabo in nadgradnjo API-ja, imenovanega BaseExternWidget, ki ga definira WinCC OA. Kakor sam WinCC OA, je tudi njegov API napisan v programskem jeziku C++, zato ga uporabimo. Pri implementaciji vmesnika BaseExternWidget dodamo tudi različne funkcije za manipulacijo aplikacije preko kontrolnih skript.

4.2.4 C++ in Visual C++

Za implementacijo uporabimo programski jezik C++, ne vemo pa še, kateri standard in razvojno okolje bomo uporabili. V analizi sistema WinCC OA ugotovimo, da je WinCC OA implementiran v integriranem razvojnem okolju (angl. *integrated development environment*, okr. *IDE*) Visual C++ 2010 [34]. Visual C++ je last podjetja Microsoft in nudi različno pokritost C++ standardov v posameznih različicah IDE Visual C++. Ker implementacija standarda uporabljena za implementacijo WinCC OA (Visual C++ 2010) ne pokriva standarda C++11 v celoti, se uporabi standard C++98, ki je v Visual C++ 2010 implementiran v celoti. Poleg standarda C++ izbiramo tudi IDE, eno od različic Visual C++, od verzije 2010 naprej. Pri implementaciji razširitev za WinCC OA z uporabo priloženega API-ja je priložene knjižnice z aplikacijo potrebno dinamično povezati. Ker so knjižnice WinCC OA zgrajene in povezane z orodjem Visual C++ 2010, jih novejše različice Visual C++ ne znajo uporabljati, zato se najprej odločimo za uporabo Visual C++ 2010. Kasneje ugotovimo, da naša aplikacija ne uporablja priloženih WinCC OA knjižnic, zato jo implementiramo v novejši različici Visual C++ 2012. S tem si poenostavimo delo, saj ima ta različica Visual C++ privzeto vključene 64 bitni prevajalnike, ki jih ni potrebno namestiti ločeno s posodobitvenim paketom, kot pri Visual C++ 2010.

Boost

Ob uporabi standarda C++98 smo prikrajšani uporabe novosti, ki nam jih ponujajo novejši standardi (C++03, C++11, C++14 in C++17). Te novosti med drugim vključujejo strukture za avtomatsko upravljanje s pomnilnikom, niti in shinronizacijske mehanizme. Odločimo se za skupino zunanjih knjižnic C++ imenovanih boost [2], različice 1.58, ki so namenjene široki uporabi in strmijo k uporabnosti v celotnem spektru vrst aplikacij in ponujajo implementacijo uveljavljenih praks.

4.2.5 Qwt

Prikazovanje velike količine podatkov v realnem času (REQ-6) v obliki krivulj v dvodimenzionalni ravnini (REQ-3) je zahtevna naloga, saj vključuje neprestano izrisovanje točk in njihovo brisanje, procesiranje uporabniških interakcij kot so prikazovanje ali skrivanje krivulj z legendo (REQ-11) in pomikanje po dvodimenzionalnem prostoru v vse smeri s spreminjanjem ločljivosti (REQ-13).

Obstaja knjižnica Qt Widgets za tehnične aplikacije [33] (angl. *Qt Widgets for technical applications*, okr. *Qwt*), ki zadovoljuje naštetim zahtevam. Knjižnica Qwt temelji na aplikacijskem ogrodju Qt. Vsebuje komponente grafičnega vmesnika in razrede, ki so primarno namenjenim tehničnim aplikacijam. Poleg glavnega ogrodja namenjenega prikazovanju dvodimenzionalnih grafov (razred QwtPlot), ponuja Qwt še mnogo drugih komponent kot so skale (razred QwtScale), legende (razred QwtLegend), kompasi, termometri in gumbi. Qwt skrbi za obdelavo in prikazovanje velikih količin podatkov z ustreznimi algoritmi. Aplikacijo tako glede na vizualni izgled, ki ga bo ponujala končnemu uporabniku, poimenujemo grafični prikazovalnik podatkov v realnem času.

4.2.6 LOG4CXX

Pri razvoju programske opreme se srečuamo s problemi oziroma tako imenovanimi hrošči (angl. *bugs*) v programski kodi, ki jih želimo čim hitreje najti in odpraviti. Najti problem včasih ni tako trivialno opravilo, kot se zdi, saj morda ne razumemo programske kode v celoti ali pa imamo opravka z večimi nitmi. Prvi klic na pomoč so v tem primeru razhroščevalniki (angl. *debuggers*), a tudi ti niso zmeraj na voljo. Iskanje problema si lahko olajšamo z enostavnim pristopom, z medoto bisekcije z uporabo beleženja [35, 15] (angl. *logging*). Uporaba beleženja je v taki ali drugačni obliki zelo pogosta praksa iskanja problemov v programski kodi. Poleg izpisovanja vsebine na standardni izhod obstajajo samostojne rešitve v obliki knjižnic, ki poskrbijo za samo

beleženje in težave, na katere lahko pri tem naletimo. Beleženje ni uporabno samo za reševanje problemov v programski kodi, saj je njegova ključna naloga beleženje pomembnih dogodkov, ki se dogajajo v aplikaciji, kar nam lahko pomaga pri doseganju zanesljivosti in varnosti.
Pri implementaciji prikazovalnika bo beleženje uporabljeno pri izdelavi začetnega delujočega ogrodja, kjer bodo vključeni in združeni vsi že omenjeni API-ji. Beleženje bomo v metodi bisekcije uporabili za iskanje problemov v programski kodi. Aplikacija bo beležila tudi pomembne dogodke za zagotavljanje zanesljivosti. To nam bo omogočilo ogrodje log4cxx [1], ki je izdelano za programski jezik C++.

Poglavje 5

Načrtovanje in implementacija

V naslednjih poglavjih sta opisana procesa načrtovanja in implementacije prikazovalnika. Pred implementacije najprej izdelamo arhitekturni načrt programske opreme [30] (angl. *software architecture*), ki mu sledimo pri implementaciji. Izdelava arhitekture nam omogoča sprejemanje odločitev, glede na kvalitativne zahteve aplikacije kot so hitrost, zanesljivost in skalabilnost. Arhitekturo izdelamo na podlagi ugotovitev in omejitev iz faze analize zahtev. Omogoča nam izvedbo osnovne analize aplikacije še preden jo izdelamo. Tako lahko uvidimo morebitne težave še preden pride do njih pri procesu implementacije.
Pri implementaciji se držimo izdelanega arhitekturnega načrta, zraven pa opišemo še, kako je aplikacija implementirana oziroma kako deluje. To želimo opisati na način, ki bo bralcu najbolj jasen in razumljiv, zato dokumentacijo razdelimo na sklope. Ti se med seboj izključujejo, da ne opisujemo vseh stvari naenkrat in da pri dokumentiranju držimo rdečo nit, s čimer dosežemo jasnost. Pri opisovanju uporabimo naslednje sklope:

- Model niti (angl. *thread model*), ki našteje vse sodelujoče niti, opiše njihovo delo in medsebojna prepletanja;
- Podatkovni model (angl. *data model*), ki opisuje uporabljene podatkovne strukture, njihove medsebojne relacije in način upravljanja s pomnilnikom (angl. *memory handling*);

- Model dogodkov (angl. *event model*), ki opisuje način rokovanja z dogodki v aplikaciji;
- Vmesniki (angl. *interfaces*), kjer so opisani vsi uporabljeni vmesniki.

5.1 Arhitektura

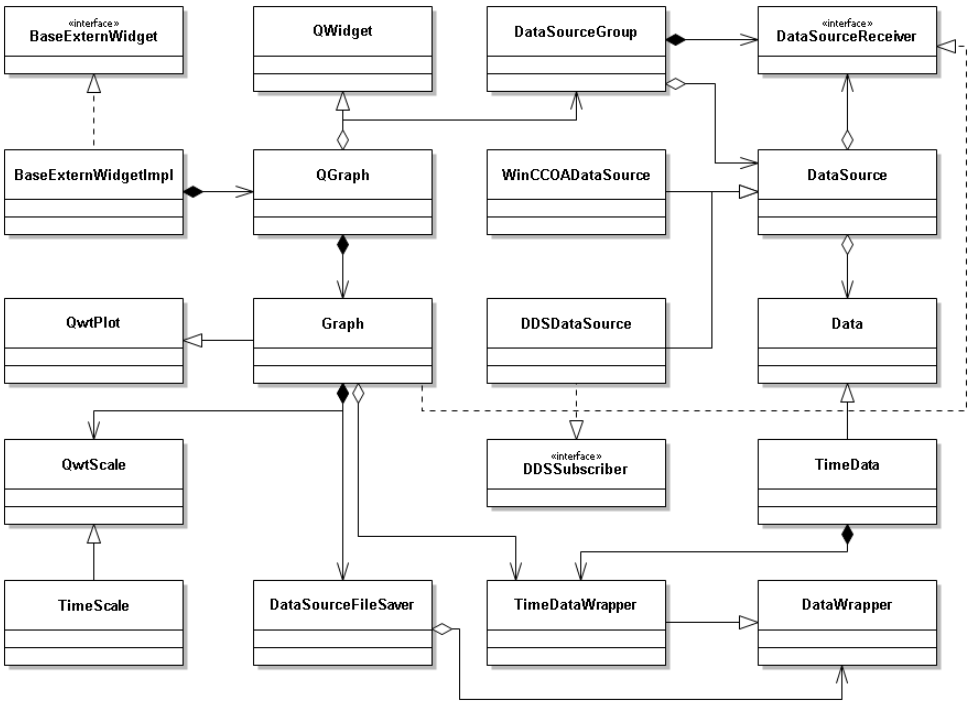
Arhitekturo aplikacije ponazorimo vizualno, zato jo opišemo diagrami v jeziku UML [32] (angl. *unified modeling language*). UML je modelarski jezik, ki se uporablja predvsem na področju računalništva za vizualizacijo sistemske arhitekture. Sistemsko arhitekturo lahko opišemo s strukturnimi ali vedenjskimi diagrami. Arhitekturo prikazovalnika podatkov opišemo s strukturnim diagramom, natančneje z razrednim diagramom. Vedenjskih diagramov ne uporabimo, ker njihovo vsebino opisuje model dogodkov. Razredni diagram je podvrsta strukturnih diagramov. Prikazujejo elemente kot so razredi, vmesniki in asociacije, hkrati pa ponazorijo tudi relacije med njimi. S tem dobimo objektno orientiran pogled na sistem, ki se uporablja v razvojne namene. Na sliki razrednega diagrama 5.1 so prikazani naslednji razredi:

- **BaseExternWidget** definira vmesnik za komunikacijo med kontrolnimi skriptami in Qt aplikacijami;
- **BaseExternWidgetImpl** implementira vmesnik BaseExternWidget.
- **QWidget** je Qtjev razred, ki implementira osnovne funkcionalnosti Qtjevih objektov. Ena od the funkcionalnosti je sistem signalov in rež;
- **QGraph** razširja razred QWidget in implementira dodatne lastnosti sistema Q_PROPERTY;
- **QwtPlot** je Qwt-jev razred, ki omogoča prikazovanje podatkov na dvodimenzionalnem grafu;
- **Graph** razširja razred QwtPlot in implementira dodatne metode za rokovanje z dogodki, ki spreminjajo način prikaza podatkov na prikazo-

valniku. Razred Graph med drugim tudi preobložuje virtualne metode razreda QwtPlot, da prikazuje podatke pridobljene s strani storitve DDS;

- **QwtScale** je Qwt-jev razred, ki ga za izrisovanje koordinatnih osi uporablja razred QwtPlot. Na koordinatnih oseh prikazuje realna števila;
- **TimeScale** razširja razred QwtScale in namesto realnih števil na koordinatni osi prikazuje časovni žig (angl. *timestamp*);
- **DataSource** definira vmesnik za impementacijo poljubnih podatkovnih izvorov in hkrati implementira mehanizme distribucije podatkov poslušalcem (v tem primeru je to objekt razreda Graph);
- **DDSSubscriber** implementira protokol za komunikacijo s storitvijo DDS in definira vmesnik za implementacijo prejemnikov (naročnikov) podatkov od storitve DDS;
- **DDSDataSource** razširja razred DataSource in implementira vmesnik DDSSubscriber za prejemanje in obdelavo podatkov prejetih od storitve DDS;
- **WinCCOADataSource** razširja razred DataSource in omogoča prejemanje podatkov iz kontrolnih skript;
- **Data** definira osnovne podatkovne strukture za shranjevanje podatkov prejetih od razreda DataSource in implementira osnovne metode za rokovanje z njimi;
- **TimeData** razširja razred Data za podatke, kjer so vrednosti točk na abscisni osi predstavljene s časovnimi žigi, na ordinatni osi pa z realnimi vrednostmi;
- **DataWrapper** implementira osnovne metode in podatkovne strukture, ki jih prikazuje razred QwtPlot;

- **TimeDataWrapper** razširja DataWrapper in skrbi za pretvorbo podatkov razreda TimeData v podatke, ki jih prikazuje QwtPlot;
- **DataSourceReceiver** definira vmesnik poslušalca, ki omogoča prejemanje podatkov od objektov razreda DataSource. Ta vmesnik implementira razred Graph;
- **DataSourceGroup** skrbi za upravljanje instanc naslednikov razreda DataSource (DDSDataSource in WinCCOADataSource);
- **DataSourceFileSaver** skrbi za shranjevanje prikazanih podatkov na trdi disk.



Slika 5.1: Razredni diagram grafičnega prikazovalnika podatkov.

5.2 Model niti

Model večnitnega programiranja lahko uporabljamo pri delu z večopravilnimi operacijskimi sistemi. Niti so del procesa in si delijo njegove vire. Izvajanje posameznih niti je neodvisno od drugih. Z večnitnim programiranjem lahko dosežemo odzivnost aplikacij, hitrejše izvajanje, manjšo porabo virov in paralelizacijo dela. Večnitno programiranje od nas zahteva dodatno delo, saj je za želeno delovanje potrebno poskrbeti preprečiti sočasne dostope niti do istih podatkov. Na kratko, niti morajo biti sinhronizirane za želeno delovanje. To dosežemo s postavitvijo sinhronizacijskih konstruktov v programsko kodo. Sinhronizacijski konstrukti kot so ključavnice (angl. *locks*), prepreke (angl. *barriers*) in semaforji (angl. *semaphores*) poskrbijo, da v določen del kode vstopa največ ena ali poljubno število niti hkrati ali da se izvajanje programa ne nadaljuje, dokler vse niti ne dosežejo določene točke v programski kodi. Pri uporabi sinhronizacijskih konstruktov tvegamo pojav dogodka, kjer vse niti čakajo ena na drugo, program pa se neha izvajati (angl. *dead lock*). Pri večnitnem programiranju je pomembno vedeti tudi, da se ob končanju niti zaradi napake ustavijo še vse ostale niti skupaj s procesom.

Sodelujoče niti v aplikaciji so:

- glavna GUI nit,
- nit za prejemanje podatkov od storitve DDS.
- niti za izvajanje kontrolnih skript,

5.2.1 Glavna GUI nit

Glavna GUI nit alocira spomin in ustvari objekt razreda BaseExternWidgetImpl. Po izhodu iz konstruktorja se nit ne zaključi temveč preide v neskončno zanko razreda QGraph imenovano *event loop*. *Event loop* je programski konstrukt, ki v neskončni zanki čaka na dogodke in jih procesira, dokler se aplikacija ne konča.

Event loop predstavlja ključno prednost, ko imamo opravka s prepletanjem niti. Ostale niti ob proženju različnih dogodkov delegirajo delo glavni GUI niti. S tem se sinhronizaciji med nitmi v večini primerih izognemo. S tem prihranimo čas implementacije in zmanjšamo kompleksnost kode.

Glavna GUI nit se imenuje glavna, ker se v njej izvajajo vse operacije, ki so neposredno povezane s prikazovanjem krivulj na prikazovalniku. Za ostale operacije, kot so prejemanje podatkov od storitve DDS in proženje dogodkov iz kontrolnih skript, skrbijo ostale niti.

5.2.2 Nit za prejemanje podatkov od storitve DDS

Kot naročniki na podatke od storitve DDS, želimo biti obveščeni vsakič, ko pod našo naročnino prispejo novi podatki. Ne želimo si, da bi neprestano preverjali ali so novi podatki že prispeli ali ne. Komunikacijo s storitvijo DDS upravlja vmesnik DDSSubscriber. Ta s storitvijo DDS komunicira preko vtičnika (angl. *socket*), z uporabo svojega protokola TCP/IP. Vtičniki v privzetem delovanju čakajo na nove podatke dokler ti ne pridejo, česar si ne želimo. Zato vmesnik storitve DDS to počne v dodatni niti, ob prejetju podatkov pa proži povratni klic.

5.2.3 Niti za izvajanje kontrolnih skript

Managerji WinCC OA lahko izvajajo več kontrolnih skript hkrati. To pomeni, da do BaseExternWidget vmesnika, ki ga implementira razred BaseExternWidgetImpl, lahko dostopa več niti hkrati. Kontrolne skripte se lahko prožijo kot povratni klici na spremembe v podatkovni strukturi WinCC OA (spremembe vrednosti DPE) ali ob uporabniških interakcijah na GUI-ju, kot so pritiski miškinih gumbov in tipk na tipkovnici. Dostopi niti do prikazovalnika preko vmesnika BaseExternWidget so torej sinhronizirani, kar pomeni, da ga lahko uporabi ena nit naenkrat.

5.3 Podatkovni Model

Podatkovni model opisuje uporabljene podatkovne strukture in način upravljanja s pomnilnikom. Pri izvajanju programa dobijo spremenljivke in objekti svoj del spomina, v katerem živijo. Izbiramo lahko med dvema vrstama spomina: To sta sklad (angl. *stack*), ki se upravlja avtomatsko, in kopica (angl. *heap*), s katero upravljamo sami. Ker smo odgovorni za uporabo (alociranje, inicializiranje, sproščanje) spomina na kopici, je pomembno, da po uporabi določenega kosa pomnilnika, tega vrnemo nazaj v lastništvo operacijskemu sistemu. Pri upravljanju spomina iz kopice lahko pride do lukenj v pomnilniku (angl. *memory leaks*), ko uporabnik prevzame del pomnilnika v svoje lastništvo in ga nikoli ne vrne. Tako operacijski sistemi čez čas izgublja lastništvo nad razpoložljivim spominom, kar lahko pripelje do zrušitve sistema.

Primer uporabe spomina na kopici je alociranje in inicializiranje objekta razreda BaseExternWidgetImpl, ki ga izvede WinCC OA in kasneje poskrbi tudi za sproščanje spomina. Drug primer je alokacija pomnilnika za objekt razreda TimeData, ki ga ustvari in s podatki, prejetimi od storitve DDS, napolni DDSSubscriber. DDSSubscriber tudi sprosti ta kos pomnilnika, ko podatke posreduje poslušalcem, in ga tako vrne nazaj v lastništvo operacijskemu sistemu.

V prikazovalniku spomin iz kopice upravljamo na 3 načine:

- z uporabo Qtjevega mehanizma preko relacij staršev in otrok
- z uporabo pametnih kazalcev (angl. *smart pointers*)
- s posrednim uporabljanjem sistemskih klicev za dodelitev in sproščanje spomina

5.3.1 Qt mehanizem preko relacij staršev in otrok

Objekti razreda QtObject so organizirani v objektnih drevesih [20] (angl. *object trees*), ki sestojijo iz relacij staršev in otrok. Ob inicializaciji novega objekta razreda QtObject njegovega naslednika na kopici lahko tega definiramo kot otroka drugega objekta (staršu).
To nam omogoča, da se ob končanju starša, sprosti tudi pomnilnik njegovih otrok. V primeru prikazovalnika bi se sprostil tudi spomin vseh gumbov, koordinatnih osi, prikaznih polj in drugih elementov. Sprostiti je mogoče tudi samo otrokov pomnilnik, ob čemer se otrok odstrani iz starševega seznama otrok v izogib dvakratnega sproščanja istega dela pomnilnika.

5.3.2 Pametni kazalci

Pametni kazalci ponujajo način upravljanja s viri (pomnilnikom), ki temelji na obsegu življenjskega cikla objekta [27] (angl. *scope based resource management*, okr. *SBRM*). Pametni kazalec je programska struktura, ki alocira pomnilnik v konstruktorju in ga sprošča v destruktorju. Pri implementaciji uporabimo pametni kazalec, ki ovije ustvarjen objekt in poskrbi za sproščanje alociranega pomnilnika. Prikazovalnik uporablja pametne kazalce za upravljanje s pomnilnikom vseh ne Qtjevih objektov razen objektov razredov, ki so potomci razreda DataWrapper. Prav tako to ne velja za objekte razreda TimeData, ki jih ne ustvari DDSSubscriber temveč jih naredi nit WinCC OA.

5.3.3 Sistemski klici

Upravljanje s pomnilnikom na kopici z uporabo Qtjevega mehanizma preko relacij staršev in otrok in uporabo pametnih kazalcev olajša ročno upravljanje s pomnilnikom (ki se v ozadju še vedno dogaja) z uporabo sistemskih klicev. Osnoven način za ročno upravljanje pomnilnika v programskem jeziku C++ omogočata rezervirani besedi *new*, ki alocira in inicializira objekt določenega razreda in *delete*, ki sprosti pomnilnik iz uporabe. Z ročnim

načinom upravljanja s pomnilnikom najhitrje pride do lukenj v pomnilniku, saj z njim upravljamo sami in smo kot ljudje manj zanesljivi kot stroji. V tem pogledu sta zgoraj omenjena načina (Qt skozi relacije staršev in otrok, pametni kazalci) v prednosti, hkrati pa prinašata tudi svoje omejitve, na primer: Uporaba pametnih kazalcev v cikličnih strukturah, ki kažejo same nase, povzroči luknje v pomnilniku, saj zmeraj obstaja nekdo, ki uporablja tisti objekt (objekt uporablja sam sebe).

Prikazovalnik ročno ustvari objekte razreda TimeWrapper, ki kopira podatke iz TimeData v svoj kos pomnilnika, preden jih zbriše nit za prejemanje podatkov od storitve DDS. Tako je objekt TimeWrapper v lastništvu glavne GUI niti, ki ga zbriše, ko podatkov ne potrebuje več. Ročni način uporabljamo tudi pri ustvarjanju objekta razreda TimeData, ko tega ne naredi nit za prejemanje podatkov od storitve DDS, temveč mi, ko prejmemo podatke iz kontrolnih skript.

5.3.4 Podatkovne strukture

TimeData

Objekti razreda TimeData, ki jo prejmemo od niti za prejemanje podatkov od storitve DDS, je sestavljena iz dveh dvodimenzionalnih polj, v katerih so shranjeni podatki o x in y kooridnatah točk poljubnega števila krivulj. TimeData vsebuje tudi asociativno polje, kjer hrani vrednosti o različnih dodatnih atributih kot sta imeni x in y osi, barve krivulj in podobno.

TimeDataWrapper

TimeDataWrapper pretvori podatke iz strukture TimeData, da jih na prikazovalniku lahko prikaže objekt razreda Graph.

5.4 Model dogodkov

Model dogodkov navaja in opisuje načine rokovanja z dogodki, ki se prožijo v aplikaciji.

5.4.1 Qt Event Loop

Programski konstrukt *event loop* implementira vsak naslednjik Qtjevega razreda QObject. Gre za neskončno zanko, ki neprestano čaka na dogodke, ki se nabirejo v dogodkovni vrsti (angl. *event queue*) in jih procesira, dokler se aplikacija ne konča. Dogodke lahko razdelimo v tri kategorije, glede na to, kako so odposlani (angl. *dispatched*):

- Spontani dogodki (angl. *spontaneous events*), ki jih generira operacijski sistem ali upravljalec oken (angl. *window manager*). Nabirajo se v dogodkovni vrsti, kjer jih neskončna zanka procesira enega za drugim. V to kategorijo spadajo uporabniške interakcije z aplikacijo (pritiski gumbov na miški in tipkovnici) ob uporabi vmesnika Q_PROPERTY ali navigaciji po prikazovalniku in proženje povratnega klica ob izteku časovnikov;

- Objavljeni dogodki (angl. *posted events*), ki jih generira Qt ali aplikacija. Prav tako kot spontani dogodki se tudi ti nabirajo se v dogodkovni vrsti, kjer jih neskončna zanka procesira enega za drugim. To so dogodki, ki se prožijo ob uporabi vmesnika BaseExternWidget in ob razpošiljanju podatkov prejetih od storitve DDS;

- Poslani dogodki (angl. *sent events*), ki jih generira Qt ali aplikacija. Ti se ne nabirajo v dogodkovni vrsti temveč so poslani ciljnemu objektu. Primer take operacije skrivanje in prikazovanje gumbov ob kliku na gumb za skrivanje in razširjanje orodne vrstice.

Neskončna zanka (*Event loop*) teče v glavni GUI niti.

5.4.2 Prejemanje podatkov od storitve DDS

Nit za prejemanje podatkov od storitve DDS posreduje nov paket podatkov (objekt razreda TimeData) objektu razreda DDSDataSource, ki ga ustvarimo z uporabo BaseExternWidget vmesnika. Preko povratnega klica implementiranega v razredu DDSDataSource paket pošljemo do objekta razreda Graph. Tam se ustvari kopija podatkov (objekt razreda TimeDataWrapper), v dogodkovno vrsto glavne GUI niti pa se doda dogodek. Za nit za prejemanje podatkov od storitve DDS sprosti objekt razreda TimeData in se vrne k čakanju na nove podatke.

5.4.3 Prejemanje podatkov iz kontrolnih skript

Podatke, prejete iz kontrolnih skript preko vmesnika BaseExternWidget, najprej pretvorimo v objekt razreda TimeData in ga posredujemo objektu razreda WinCCOADataSource. Če objekta razreda WinCCOADataSource še ne obstaja, ga ustvarimo. Ta podatke pošlje objektu razreda Graph, ki z njimi rokuje na enak način kot nit za prejemanje podatkov od storitve DDS.

5.4.4 Izrisovanje podatkov na prikazovalniku

Izrisovanje podatkov na prikazovalniku je glavni del aplikacije, ki jo izvaja glavna GUI nit. Ko glavna GUI nit procesira nov dogodek iz dogodkovne vrste, z novim paketom podatkov (objekt razreda TimeDataWrapper), najprej prebere in uporabi njegove dodatne lastnosti kot so ime koordinatnih osi, imena krivulj in druge. Za tem se novi podatki pripnejo obstoječim ali pa jih zamenjajo. Preden glavna GUI nit zaključi z rokovanjem tega dogodka sprosti še podatke, ki jih ne potrebuje več, nato pa se vrne v dogodkovno zanko.

Ker je izrisovanje podatkov zahtevna operacija, tega ne želimo izvajati vsakič ko pride nov paket, saj pri večjih količinah novih podatkov uporabnik tega ne bi zaznal, prikazovalnik pa bi tako lahko v realnem času prikazoval veliko manj podatkov. Izrisovanje podatkov se zato izvaja vsakih 100 milise-

kund, s čimer pridobimo na zmogljivosti. Za proženje ponovnega izrisa skrbi časovnik, ki se proži vsakih 100 milisekund.

5.4.5 Beleženje izjem

Izjeme, kot sta prekinitev povezave do storitve DDS in napaka pri prenosu podatkov, beležimo v datoteko z uporabo storitve LOG4CXX, ki jo konfiguriramo preko vmesnika BaseExternWidget.

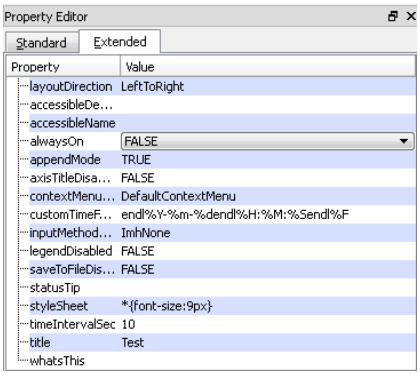
5.5 Vmesniki

5.5.1 Vmesnik Q_PROPERTY

Vmesnik Q_PROPERTY definira lastnosti prikazovalnika, ki se ne spreminjajo v času prikazovanja podatkov na prikazovalniku. Določimo jih pred začetkom prejemanja podatkov. Nastavimo jih v WinCC OA-jevem orodju gedi, ki zapomni nastavljene vrednosti lastnosti. Tabela 5.1 našteva in opisuje lastnosti vmesnika Q_PROPERTY, slika 5.2 pa prikazuje njegov grafični vmesnik v orodju gedi.

Ime lastnosti	Opis
alwaysOn	TRUE - prikazovalnik začne s prejemanjem takoj po inicializaciji; FALSE - prikazovalnik ne začne prejemati, dokler uporabnik ne določi začetka.
title	Določi naslov grafa.
saveToFileDisabled	TRUE - brez shranjevanja v datoteko; FALSE - shranjevanje v datoteko je omogočeno.
appendMode	TRUE - novi podatki se pripenjajo starim; FALSE - novi podatki zamenjajo stare.
timeIntervalSec	Določa maksimalno starost podatkov (v sekundah) za način appendMode = TRUE.
axisTitleDisabled	TRUE - naslovi koordinatnih osi niso vidni; FALSE - naslovi koordinatnih osi so vidni.
legendDisabled	TRUE - legenda krivulj je skrita; FALSE - legenda krivulj je vidna.
customTimestampFormat	Določa format zapisa časovnih žigov na abscisni osi.

Tabela 5.1: Seznam lastnosti vmesnika Q_PROPERTY in njihov opis.



Slika 5.2: Grafični vmesnik Q_PROPERTY v orodju gedi.

5.5.2 Vmesnik BaseExternWidget

Vmesnik BaseExternWidget omogoča manipulacijo prikazovalnika iz kontrolnih skript. Vmesnik sestavljajo funkcije, do katerih dostopamo preko vstopne metode *invokeMethod*, ki nato izvede izbrano funkcijo. Ta način definira vmesnik BaseExternWidget kot ozki vmesnik (angl. *narrow interface*). Tabela 5.2 našteva in opisuje funkcije vmesnika BaseExternWidget, ki ga implementira razred BaseExternWidgetImpl.

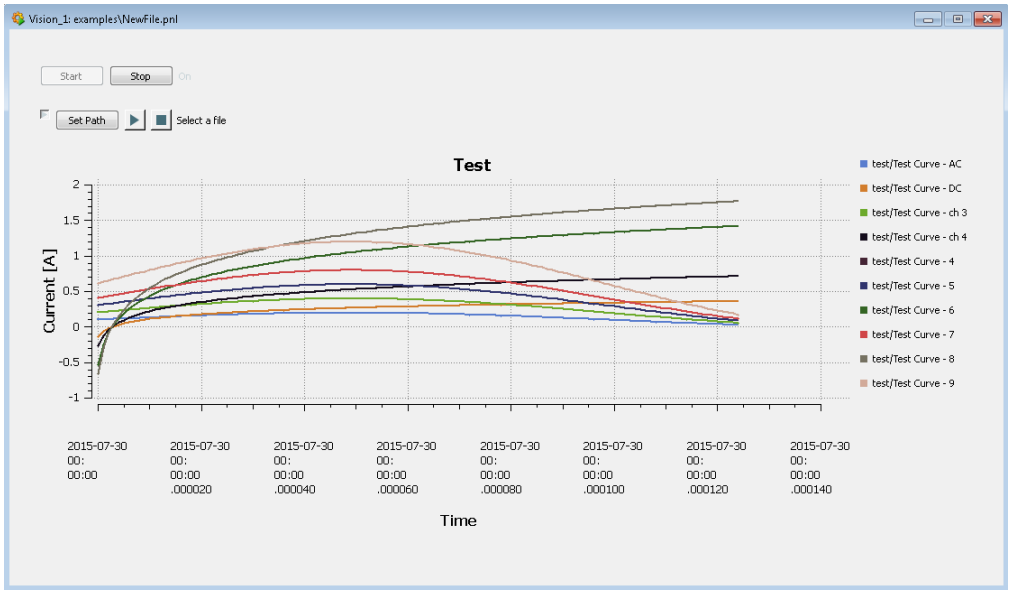
Ime funkcije	**Opis**
setDDSIp	Nastavi IP naslov strežnika storitve DDS.
setSubscriptions	Sklene naročnine s storitvijo DDS.
setLogConfFile	Poda pot do konfiguracijske datoteke za konfiguracijo storitve beleženja dogodkov.
setSavePath	Nastavi pot za shranjevanje podatkov v datoteko.
setLang	Nastavi jezik, ki ga uporablja prikazovalnik.
fillData	Prikaže podatke iz kontrolnih skript na grafu.

Tabela 5.2: Seznam in opis funkcij vmesnika BaseExternWidget.

5.5.3 Uporabniški grafični vmesnik

GUI prikazuje podatke v obliki krivulj na dvodimenzionalni ravnini. Dvodimenzionalna ravnina je prikazana kot graf s koordinatnima osema, kjer y os predstavlja numerične vrednosti točk, x os pa y koordinato umesti v določeno časovno točko. GUI nam omogoča še:

- spreminjanje ločljivosti prikazovalnika,
- pomikanje po koordinatnih oseh v poljubnih smereh,
- skrivanje in prikzovanje krivulj preko legende,
- nadzor nad prejemanem podatkov,
- nadzor nad shranjevanjem prikazanih podatkov v datoteko.



Slika 5.3: Izgled grafičnega prikazovalnika podatkov v realnem času.

Poglavje 6

Testiranje

S testiranjem preverimo delovanje aplikacije glede na specificirane zahteve. Za prikazovalnik podatkov nas zanimajo predvsem zmogljivostni testi, ker je njegova glavna funkcionalnost grafično prikazovanje podatkov v realnem času. Z zmogljivostnimi testi preverimo koliko podatkov je prikazovalnik zmožen prikazati v realnem času in kako se grafični vmesnik odziva pri velikih obremenitvah.
Ob testiranju aplikacijo izpostavimo velikim obremenitvam, ki so večje od specificiranih. Tako preverimo, da se aplikacija zares obnaša, kot smo predvideli.
Zelo pomembno je, da imamo aplikacijo pod kontrolo. To posledično pomeni, da znamo teste ponoviti in da vsakič dobimo enake rezultate. Če želimo to doseči moramo test vsakič izvesti na popolnoma enak način, zato za vsak test specificiramo naslednje:

- kaj testiramo,
- kaj merimo,
- kako merimo,
- kakšna so naša bremena,
- kaj je rezultat testa,

- pod kakšnimi pogoji testiramo,
- procedura testa,
- izmerjen rezultat.

6.1 Test: Prikazovanje v realnem času

Kaj testiramo?

Zanima nas maksimalno število točk na sekundo, ki jih lahko prikazovalnik prikazuje v realnem času.

Kaj merimo?

Merimo število točk (točka ima koordinati x in y) na sekundo (enota pts/s), ki jih prejemamo od storitve DDS. Obe koordinati sta predstavljeni v 64 bitnem zapisu, tako da njuna skupna velikost znaša 16 bajtov.

Kako merimo?

Prikazovalnik izrisuje podatke v realnem času, če preneha z izrisovanjem takoj, ko prekinemo pošiljanje podatkov preko storitve DDS.

Breme

Pošiljatelj pošilja bremena različnih velikosti:

- 100K pts/s,
- 1M pts/s,
- 10M pts/s,
- 100M pts/s.

Kaj je rezultat testa?

Rezultat testa je maksimalno število točk na sekundo, ki jih prikazovalnik prikaže v realnem času.

Pogoji

Za test uporabimo 2 računalnika. Zanima nas samo zmogljivost centralne procesne enote (CPE), ki opravlja vso delo.
Računalnik, na katerem teče prikazovalnik, ima CPE model Intel® Core™ i7-3520M, računalnik, iz katerega pošiljamo podatke, pa Intel® Core™ 2 DUO P8700. Za test uporabljamo mrežo s pretokom 100 Mb/s.
Prikazovalnik prejema podatke samodejno in uporablja način, ki zamenjuje obstoječe podatke z novimi vsakič, ko ti pridejo. Podatkov ne shranjujemo v datoteko.

Procedura

1. Test začnemo z najmanjšim bremenom;
2. Poženemo storitev DDS in začnemo s pošiljanjem podatkov na drugem računalniku;
3. Poženemo prikazovalnik podatkov s pomočjo WinCC OA na prvem računalniku;
4. Počakamo 20 sekund in prekinemo pošiljanje podatkov;
5. Test ponovimo za vsa bremena ali dokler prikazovalnik prikazuje podatke v realnem času.

Izmerjen rezultat

Izmerjen rezultat znaša 100K pts/s. Razlog, da rezultat ni višji, je omejitev prenosa podatkov preko mreže, zato višjih bremen ni bilo mogoče testirati.

6.2 Test: Odzivnost prikazovalnika

Kaj testiramo?

Zanima nas odzivnost uporabniškega vmesnika med prikazovanjem različnega števila točk a prikazovalniku.

Kaj merimo?

Merimo število prikazanih točk (enota pts) in število milisekund (enota ms), ki jih preteče po spremembi ločljivost na prikazovalniku za 1 stopnjo.

Kako merimo?

Ločljivost prikazovalnika spremenimo za 1 stopnjo s pomikom miškinega kolesa naprej ali nazaj za en korak, miškin kurzor pa se nahaja nad prikazovalnikom. Čas začnemo meriti takoj po premiku miškinega kolesa in končamo, ko grafični vmesnik ponovno izriše podatke s spremenjeno ločljivostjo.

Breme

Pošiljatelj pošilja bremena različnih velikosti:

- 100K pts,
- 1M pts,
- 10M pts,
- 100M pts.

Kaj je rezultat testa?

Rezultat testa je razpredelnica, ki za vsako breme vsebuje izmerjeni čas odzivnosti.

Pogoji

Za test uporabimo 2 računalnika. Zanima nas samo zmogljivost centralne procesne enote (CPE), ki opravlja vso delo.
Računalnik, na katerem teče prikazovalnik, ima CPE model Intel® Core™ i7-3520M, računalnik, iz katerega pošiljamo podatke, pa Intel® Core™ 2 DUO P8700. Za test uporabljamo mrežo s pretokom 100 Mb/s.
Prikazovalnik prejema podatke samodejno in uporablja način, ki pripenja nove podatke k obstoječim. Podatkov ne shranjujemo v datoteko.

Procedura

1. Test začnemo z najmanjšim bremenom;
2. Poženemo prikazovalnik podatkov s pomočjo WinCC OA na prvem računalniku;
3. Poženemo storitev DDS, začnemo s pošiljanjem podatkov na drugem računalniku in počakamo da se breme akumulira na prikazovalniku;
4. Izmerimo odzivnost;
5. Test ponovimo za vsa bremena.

Izmerjen rezultat

Rezultati meritev so predstavljeni v tabeli 6.1.

Število točk [pts]	**Odzivni čas [ms]**
100K	0
1M	20
10M	1000
100M	16000

Tabela 6.1: Števila prikazanih točk in odzivni časi.

Literatura

[1] Apache log4cxx. https://logging.apache.org/log4cxx/. [Online; Dostopano 14. 3. 2016].

[2] boost c++ library. http://www.boost.org/. [Online; Dostopano 14.12.2015].

[3] Cascading style sheets (css). https://www.w3.org/Style/CSS/. [Online; Dostopano 14.12.2015].

[4] Qt Company. Qt. https://www.qt.io, 1994. [Online; Dostopano 14.12.2015].

[5] csv format. https://tools.ietf.org/html/rfc4180. [Online; Dostopano 14. 3. 2015].

[6] Cyclotron. https://en.wikipedia.org/wiki/Cyclotron. [Online; Dostopano 14. 3. 2015].

[7] Design pattern. https://en.wikipedia.org/wiki/Software_design_pattern. [Online; Dostopano 14. 3. 2015].

[8] Difference between scada and hmi. http://www.indusoft.com/blog/2013/04/19/what-is-the-difference-between-scada-and-hmi/. [Online; Dostopano 14. 3. 2015].

[9] a Siemens Company ETM professional control GmbH. Simatic wincc open archiecture. www.etm.at. [Online; Dostopano 14.12.2015].

[10] Eric Freeman, Elisabeth Robson, Bert Bates, and Kathy Sierra. *Head first design patterns*. "O'Reilly Media, Inc.", 2004.

[11] Johannes Gutleber, A Brett, R Moser, M Marchhart, C Torcato de Matos, and J Dedič. The medaustron accelerator control system. In *Proc. ICALEPCS*, 2011.

[12] Introspection. `https://en.wikipedia.org/wiki/Type_introspection`. [Online; Dostopano 14.12.2015].

[13] Linac or cyclotron. `http://myrrha.sckcen.be/en/Engineering/Accelerator/Linac_cyclotron`. [Online; Dostopano 14. 3. 2015].

[14] Linear particle accelerator. `https://en.wikipedia.org/wiki/Linear_particle_accelerator`. [Online; Dostopano 14. 3. 2015].

[15] Logging: The most important part of any application. `http://www.nsprogrammer.com/2013/06/logging-to-disk-most-important-part-of.html`. [Online; Dostopano 14. 3. 2016].

[16] Open platform communications. `https://en.wikipedia.org/wiki/Open_Platform_Communications`. [Online; Dostopano 14. 3. 2015].

[17] Open platform communications unified architecture. `https://en.wikipedia.org/wiki/OPC_Unified_Architecture`. [Online; Dostopano 14. 3. 2015].

[18] Particle accelerator. `https://en.wikipedia.org/wiki/Particle_accelerator`. [Online; Dostopano 14. 3. 2015].

[19] Publish-subscribe design pattern. `https://msdn.microsoft.com/en-us/library/ff649664.aspx`. [Online; Dostopano 14. 3. 2015].

[20] Qt object trees. `https://doc.qt.io/qt-5/objecttrees.html`. [Online; Dostopano 6. 6. 2016].

[21] Qt q_property system. `https://doc.qt.io/qt-4.8/properties.html`. [Online; Dostopano 14.12.2015].

[22] Qt signal-slot system. `https://doc.qt.io/qt-4.8/signalsandslots.html#signals-and-slots`. [Online; Dostopano 14.12.2015].

[23] Qt stylesheets. `https://doc.qt.io/qt-4.8/stylesheet-reference.html`. [Online; Dostopano 14.12.2015].

[24] Qt wikipedia. `https://en.wikipedia.org/wiki/Qt_(software)`. [Online; Dostopano 14.12.2015].

[25] Quadrupole magnet. `https://en.wikipedia.org/wiki/Quadrupole_magnet`. [Online; Dostopano 14. 3. 2015].

[26] Radio frequency cavities. `http://home.cern/about/engineering/radiofrequency-cavities`. [Online; Dostopano 14. 3. 2015].

[27] Scope based resource management. `http://allenchou.net/2014/10/scope-based-resource-management-raii/`. [Online; Dostopano 6. 6. 2016].

[28] J Serrano, P Alvarez, M Lipinski, T Włostowski, et al. Accelerator timing systems overview. *Proceedings of IPAC*, 2011.

[29] Sextupole magnet. `https://en.wikipedia.org/wiki/Sextupole_magnet`. [Online; Dostopano 14. 3. 2015].

[30] Software architecture. `https://en.wikipedia.org/wiki/Software_architecture`. [Online; Dostopano 31. 5. 2016].

[31] Synchrotron. `https://en.wikipedia.org/wiki/Synchrotron`. [Online; Dostopano 14. 3. 2015].

[32] Unified modeling language. `https://en.wikipedia.org/wiki/Unified_Modeling_Language`. [Online; Dostopano 31. 5. 2016].

[33] Josef Wilgen Uwe Rathmann. Qwt. http://qwt.sourceforge.net/. [Online; Dostopano 14.12.2015].

[34] Visual c++. https://en.wikipedia.org/wiki/Visual_C%2B%2B. [Online; Dostopano 14.12.2015].

[35] Why is logging important. https://czanik.blogs.balabit.com/2013/04/why-logging-is-important/. [Online; dostopano 14. 3. 2016].

[36] Herman Winick and Sebastian Doniach. *Synchrotron radiation research*. Springer Science & Business Media, 2012.